Part 1: Adapted from Astropy Tutorial On Viewing and Manipulating FITS Images

Authors

Lia Corrales, Kris Stern

Learning Goals

1. Learn how to open FITS files and load the image data
2. Learn how to visualize the image data
3. Learn about basic image math: image stacking
4. Learn how to write image data to a FITS file

Keywords

matplotlib, FITS image, table

Summary

This tutorial demonstrates the use of `astropy.utils.data` to download a data file, then uses `astropy.io.fits` to open the file, and lastly uses `matplotlib` to view the image with different color scales and stretches and to make histograms. In this tutorial we've also included a demonstration of simple image stacking.

In [0]:
```
import numpy as np

# Set up matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

from astropy.io import fits
```

The following cell is needed to download the example FITS files used here.

In [0]:
```
from astropy.utils.data import download_file
image_file = download_file('http://data.astropy.org/tutorials/FITS-image
s/HorseHead.fits', cache=True )
```

```
Downloading http://data.astropy.org/tutorials/FITS-images/HorseHead.fit
s [Done]
```

Opening FITS files and loading the image data

Let's open the FITS file to find out what it contains.

In [0]:
```
hdu_list = fits.open(image_file)
hdu_list.info()
```

```
Filename: /root/.astropy/cache/download/py3/2c9202ae878ecfcb60878ceb638
37f5f
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     161   (891, 893)   int16
  1  er.mask       1 TableHDU        25   1600R x 4C   [F6.2, F6.2, F6.
2, F6.2]
```

Generally, the image information is located in the `PRIMARY` block. The blocks are numbered and can be accessed by indexing `hdu_list`.

In [0]:
```
image_data = hdu_list[0].data
```

Our data is now stored as a 2-D numpy array. But how do we know the dimensions of the image? We can simply look at the `shape` of the array.

In [0]:
```
print(type(image_data))
print(image_data.shape)
```

```
<class 'numpy.ndarray'>
(893, 891)
```

Great! At this point, we can close the FITS file because we've stored everything we wanted to a variable.

In [0]:
```
hdu_list.close()
```

SHORTCUT

If you don't need to examine the FITS header, you can call `fits.getdata` to bypass the previous steps.

In [0]:
```
image_data = fits.getdata(image_file)
print(type(image_data))
print(image_data.shape)
```

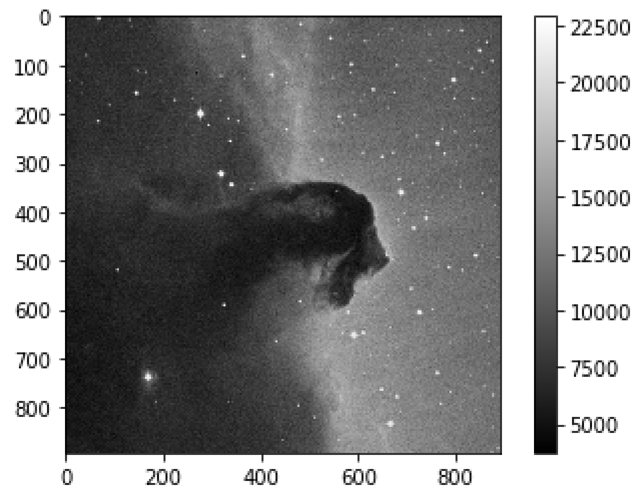
```
<class 'numpy.ndarray'>
(893, 891)
```

Viewing the image data and getting basic statistics

In [0]:
```
plt.imshow(image_data, cmap='gray')
plt.colorbar()

# To see more color maps
# http://wiki.scipy.org/Cookbook/Matplotlib/Show_colormaps
```

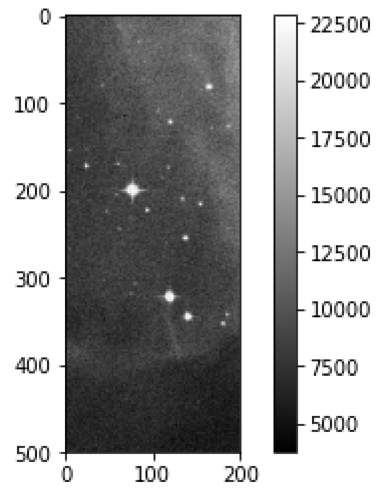
Out[0]:
```
<matplotlib.colorbar.Colorbar at 0x7fcae44d3208>
```



We can select parts of the array to plot, using indexing.

In [0]:
```
plt.imshow(image_data[0:500,200:400],cmap='gray')
plt.colorbar()
```

Out[0]:
```
<matplotlib.colorbar.Colorbar at 0x7fcae443add8>
```



Let's get some basic statistics about our image:

In [0]:
```
print('Min:', np.min(image_data))
print('Max:', np.max(image_data))
print('Mean:', np.mean(image_data))
print('Stdev:', np.std(image_data))
```

```
Min: 3759
Max: 22918
Mean: 9831.481676287574
Stdev: 3032.3927542049046
```

We can use similar numpy functions to find the location of the minimum and maximum of the data.

In [0]:
```
min_coord = print(np.where(image_data == np.amin(image_data)))
```

```
(array([116]), array([268]))
```

Verify that the data at that location does have the minimum value, found above

In [0]:
```
image_data[116,268]
```

Out[0]:
```
3759
```

In [0]:
```
max_coord = print(np.where(image_data == np.amax(image_data)))
print(max_coord)
```

```
(array([71]), array([850]))
None
```

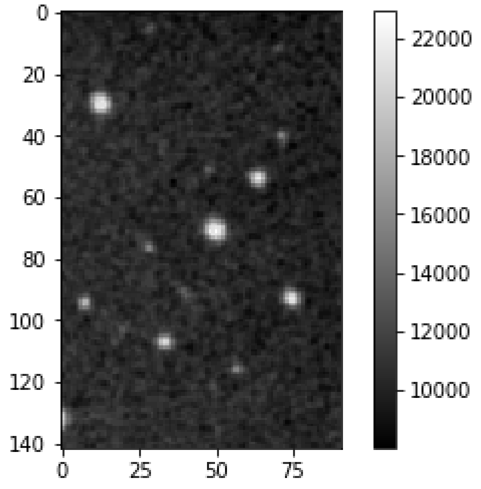
In [0]:
```
image_data[71,850]
```

Out[0]:
```
22918
```

In [0]:

In [0]:
```
plt.imshow(image_data[0:71*2,800:900],cmap='gray')
plt.colorbar()
```

Out[0]:
```
<matplotlib.colorbar.Colorbar at 0x7fcae01acd30>
```

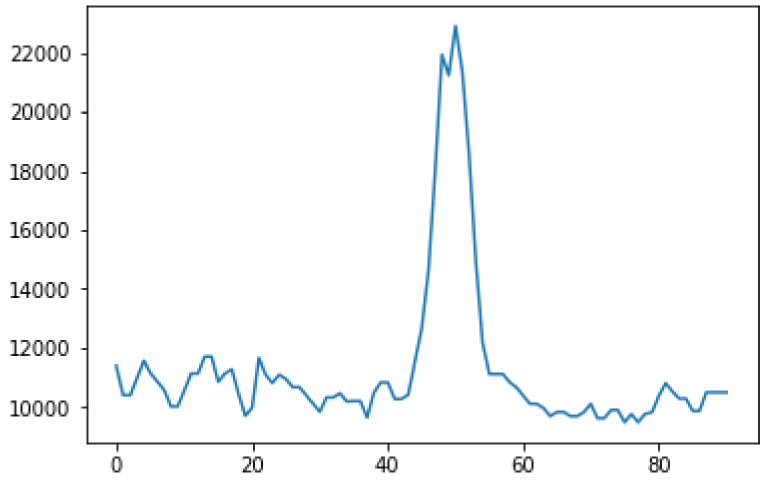


In [0]:
```
```

We can also plot a 1D "slice" of the image

In [0]:
```
plt.plot(image_data[71,800:900])
```

Out[0]:
```
[<matplotlib.lines.Line2D at 0x7fcae0123198>]
```



In [0]:
```
star_slice = image_data[71,800:900]
no_signal = star_slice[0:40]
print(np.std(no_signal))
```

```
545.057748660635
```

```
In [49]: baseline = 10000
         peak = 22000
         noise = 500
         signal = peak - baseline
         sn_ratio = signal/noise
         print('Estimated signal to noise ratio is: ', sn_ratio)

         Estimated signal to noise ratio is:  24.0
```

Plotting a histogram

To make a histogram with `matplotlib.pyplot.hist()`, we'll need to cast the data from a 2-D array to something one dimensional.

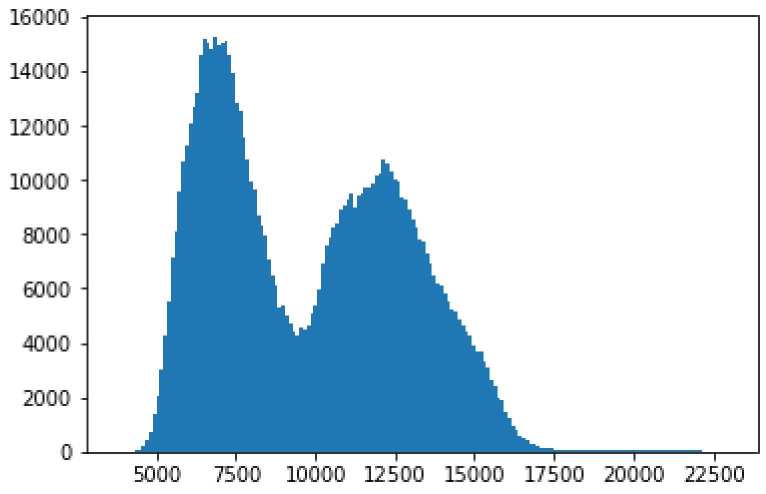
In this case, let's use the `ndarray.flatten()` to return a 1-D numpy array.

```
In [0]: print(type(image_data.flatten()))

        <class 'numpy.ndarray'>
```

```
In [0]:
```

```
In [0]: histogram = plt.hist(image_data.flatten(), bins='auto')
```


16000
14000
12000
10000
8000
6000
4000
2000
0
5000
7500
10000
12500
15000
17500
20000
22500